DDGARM: Dotlet Driven Global Alignment with Reduced Matrix

Vineetha V, Achuthsankar S. Nair
Dept. of Computational Biology & Bioinformatics, University of Kerala, Thiruvananthapuram, Kerala, India

DOI: 10.23956/ijarcsse/V7I6/0123

Abstract— Dynamic programming approach provides the most optimal alignment of sequences, but they are known to have high computational cost as well. The approach uses 2-D matrix of size \((m \times n)\), \((m & n\) being length of the sequences) to compare the sequences and hence the time and space complexity becomes \((m \times n)\). In this paper we are presenting a global alignment solution which fills only reduced number of diagonals in the 2-D matrix. Dot plot approach which is used for general exploration of sequences is used to identify the minimum amount of diagonals to be filled to get the optimal alignment. DDGARM reduces the memory and execution time significantly compared to the classic dynamic approach suggested by Needleman-Wunsch. Sequences with similar length provided a reduction of up to 90% in the number of cells to be filled and sequences having different length provided reduction up to 75% with no compromise on the alignment quality. Source code is available at: https://sourceforge.net/projects/ddgarm/

Keywords— Needleman-Wunsch Algorithm, Global Alignment, Dotplot matrix, Reduced matrix filling

I. INTRODUCTION

Sequence alignment is a fundamental problem in computational genomics. It is a way of arranging biological sequences to identify regions of similarity between them which may be a consequence of functional, structural or evolutionary relationship [1]. There are two approaches in alignment, local alignment which identifies similar segments of the sequences and global alignment which identifies overall similarity of the sequences. The widely adopted global alignment algorithm is [2], which provides optimal alignment of sequences. But this algorithm is very expensive in terms of computational space and time, which is proportional to the product of length of the two sequences. Therefore, many heuristic algorithms such as BLAST [3] and FASTA [4] are commonly used which are much faster, but do not produce optimal result.

Needleman-Wunsch (N-W) algorithm creates a 2-dimensional matrix of size \((m \times n)\), \(m\) and \(n\) being the length of the sequences involved, and then evaluates the score of each cell from the maximum of the three cells around it. Also, it keeps a pointer to the cell with maximum value. After the entire matrix is populated, it performs a trace back from cell \((m, n)\) to \((0,0)\) following the pointers to find the optimal alignment. The algorithm consumes more time to populate the matrix. Although, most of the cells are not used in the trace back, they need to be filled as they help in calculating the scores of each cell and identifying the cells needed in trace back.

In this article we present an optimal global sequence alignment which follows the N-W algorithm, but populates only minimal number of cells in the matrix.

The dot plot approach, introduced by Gibbs and McIntyre [5], provides a simple visual representation of similarity between two sequences. This approach is used in Dotlet [6], which is a platform independent implementation. Dotlet matrix is used to identify the minimum number of diagonals that need to be filled in N-W algorithm to obtain optimal alignment.

II. METHODS

Dotlet is a diagonal plot tool which is GUI driven and is platform independent. It generates a score matrix in order to identify the similarities between the sequences and plot the graph. This score matrix is used to identify the diagonals in the 2-D matrix which has the potential to be part of the trace back in identifying optimal alignment. The count of diagonals is given as in input to the modified N-W algorithm to perform global alignment. The algorithm flow is as follows:

1. Modified Dotlet is executed which accepts sequences as input
2. Algorithm identifies the cells with similar characters from score matrix and calculates the number of diagonals having cells with maximum similarity.
3. Along with the dot plot, it gives the number of diagonals to be filled as output.
4. Then the modified N-W algorithm is executed which accepts the sequences and the number of diagonals to be filled as input.
5. For sequences of similar length, algorithm populates only those cells which are part of the identified diagonals.
6. In case of sequences with different lengths, algorithm populates cells which fall within the region of difference in length between sequences in addition to the identified diagonals.
Fig. 1DDGARM Flow-chart: The basic working principle explained

Start

Execute the modified Dotlet program. This identifies the matrix cells which are having 1 as value and locates the number of diagonals which come in the alignment based on maximum occurrences of 1’s

Generate file with input sequences and also one store the number of diagonals to be filled in file

Read input sequences and number of diagonals from the file

Create 2-D matrix with input sequences as horizontal and vertical length

For each cell check if it falls under the identified diagonals?

Yes

Populate the cell with score

No

Does it fall under region of difference between sequence length?

No

When all cells are covered, perform trace back with the filled cells

End
A diagrammatic representation of alignment of sequences having same length - 5 diagonals are identified based on the presence of maximum number of 1’s in the alignment path.

A diagrammatic representation of alignment of sequences having different length - 5 diagonals are identified based on the presence of maximum number of 1’s in the alignment path. In addition to that 2 diagonals with length 10 (shorter sequence length) are filled as they fall in the region of difference between the sequences.

III. COMPLEXITY ANALYSIS

Time & space Complexity of DDGARM is described below:
When both sequences are of same length only the number of diagonals identified will be filled, the central diagonal will have the cells equal to the length of the sequences (m). To both sides of this central diagonal, we will have diagonals with number of cells decreasing by 1 from m. So, the number of cells to be filled at each side of the central diagonal can be expressed as an arithmetic progression having:

First term = (m - 1)
Number of terms = (d - 1)/2
Difference in terms = 1
nth term = (m - 1) + \left(\frac{d-1}{2}\right) - 1 ) \times 1

So, for both sides, the expression would be like
\[ \frac{d - 1}{2} \times \left( (m - 1) + \left( (m - 1) + \left( \frac{d - 1}{2} \right) - 1 \right) \right) \]

To this cells of the central diagonal needs to be added, so,
\[ \frac{d - 1}{2} \times \left( (m - 1) + \left( (m - 1) + \left( \frac{d - 1}{2} \right) - 1 \right) \right) + m \]
This can be reduced to

\[
\frac{d - 1}{2} \times [2m - 1 - \frac{d - 1}{2}] + m
\]

When sequences are having different length, we need to fill additional diagonals having number of cells equal to the length of the shorter sequence. Hence the complexity would become:

\[
\frac{d - 1}{2} \times [2m - 1 - \frac{d - 1}{2}] + m + m((n - m) - \frac{d - 1}{2})
\]

IV. RESULTS & DISCUSSION

TABLE I
COMPARATIVE VIEW OF MEMORY UTILIZATION AND EXECUTION TIME BY DDGARM AND CLASSIC N-W ALGORITHM

<table>
<thead>
<tr>
<th>Input Sequences (size in base pairs)</th>
<th>Number of Cells filled</th>
<th>Execution Time in seconds</th>
<th>% of cells filled</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Classic N-W algorithm</td>
<td>DDGARM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Number of diagonals filled</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>% of cells filled</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>333 x 333 (77% similar)</td>
<td>110889</td>
<td>10717</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9.66</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>333 X 392 (95% similar)</td>
<td>130536</td>
<td>37857</td>
<td>119 + additional for difference in length</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>29.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>333 x 433 (85% similar)</td>
<td>144189</td>
<td>38825</td>
<td>33 + additional for difference in length</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>26.93</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>14</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>1295 x 968 (46% similar)</td>
<td>1253560</td>
<td>318471</td>
<td>3 + additional for difference in length</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>25.41</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>1295 X 333 (22% similar)</td>
<td>431235</td>
<td>325236</td>
<td>29 + additional for difference in length</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>75.42</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>28</td>
</tr>
<tr>
<td>333 X 463 (87% similar)</td>
<td>154179</td>
<td>53148</td>
<td>61 + additional for difference in length</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>34.47</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>18</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>14</td>
</tr>
</tbody>
</table>

The results show that DDGARM uses very less memory compared to the classic Needleman-Wunsch algorithm. In case of sequences having same length, it provided very good improvement in the number of matrix cells filled. We can see that in some cases we got up to 90% reduction in the memory usage. Memory usage is slightly more for sequences having different length as we have to fill more number of cells to get optimal alignment.
From the results, it is evident that we cannot derive a relationship between the reduction in memory and percentage of similarity between sequences. It purely depends on the location of similar regions in the sequences. For more distant sequences the improvement in memory & execution time will be less compared to those with more similarity. For sequences having similar length, we got very good improvement in the memory usage as well as execution time as we have to fill only the identified diagonals to obtain the optimal alignment.

V. CONCLUSIONS

DDGARM provides very good improvement in the memory usage as well as execution time in finding global alignment of sequences. It addresses the major bottlenecks of Needleman-Wunsch algorithm which is well known for its accuracy but not used for longer sequences due to high computational cost. Major advantage of DDGARM is that it provides exactly the same result as we get by using classic Needleman_Wunsch algorithm, at the same time uses very less memory and execution time.

Current version of DDGARM is a prototype and hence supports only Nucleotide sequences and also involves I/O operations. Support for protein sequences and handling error scenarios are being implemented and will be released in the next version.

ACKNOWLEDGMENT

This work has been supported by State Inter University Centre of Excellence in Bioinformatics (SIUCEB), AiCADD and DBT-BIF.

REFERENCES